Proof Score Writing for QLOCK in OTS/CafeOBJ

FUTATSUGI, Kokichi
二木 厚吉
JAIST

Topics

- **Systematic construction of Proof Score for verifying the mutual exclusion property**

Theorem of constants (TC)

`INV1 |=` $\forall s \in$ `Sys` $\forall i,j \in$ `Pid.inv1(`s,i,j`)`

||def

`INV1 |= inv1(s:Sys,i:Pid,j:Pid)`

||TC

`INV1` ∪ `{op s : -> Sys}` ∪ `{ops i j : -> Pid} |= inv1(s,i,j)`

All the above three state:
for any model (or `INV1`-algebra) and
for any objects `s:Sys,i:Pid,j:Pid`,
the proposition `inv1(s,i,j)` holds.

Constants and variables: Differences

(model/interpretation is assumed to be fixed and p is assumed to contain no variables in S,E|= p)

A variable of the same name which appears in the same equation in S,E|= p denotes arbitrarily but the same object.

A variable of the same name which appears in several different equations in S,E|= p denotes any object independently, and does not necessary denote the same object.

A constant of the same name which appears in several different places in S,E|= p denotes the same object, because a constant constitutes the signature S.

Assertion and Proof Passage

proof-00.cafe

Assertion

```
INV1  U
{op s : -> Sys}  U
{ops i j : -> Pid}
  |=
inv1(s,i,j)
```

Logical Statement of stating that Specification satisfies property

~

Proof Passage

```
-- [mx]*
open INV1
 op s : -> Sys .
 ops i j : -> Pid .
-- |=
 red inv1(s,i,j) .
close
```

Logical Statement and CafeOBJ Code

If reduction part of the CafeOBJ code returns `true` then the assertion holds

If [mx]* returns `true` the verification is done, but ...

```
-- [mx]*
open INV1
 op s : -> Sys .
 ops i j : -> Pid .
-- |=
 red inv1(s,i,j) .
close
```

This assertion states that mutual exclusion property holds for any reachable states.

If this proof passage returns true the game is over. But it does not return `true`.

A name of assertion/proof-passage(p.-p.) which ends with the character `*` (like `[mx]*`) indicates that it does not return `true`. An assertion/p.-p. which return `true` is called to be effective. An effective p.p. has a name without `*` at the end.

Proof Scores and Assertion Splitting Rules (1)

The goal of verification of an assertion `A` via proof score is to get a set of assertions/p.-p.s `{A1,A2,… An}` such that:

(1) `(A1 and A2 and … and An)` implies `A`, and

(2) All the `A1,A2,…,An` are effective.

A set of assertions/p.-p.s {`A1,A2,…,An`} which satisfies (1) is called proof score (in a narrow sense) for `A` , and is also called effective proof score if it also satisfies (2).

Proof Scores and Assertion Splitting Rules (2)

For constructing proof scores, several kinds of **assertion splitting rules** of the form:

`(Ai1 and Ai2 and … and Ain)` implies `Ai`

are used. An assetion splitting rule is also written as:

`{Ai1,Ai2, … Ain} implies Ai .`

If `n` = 1 the assertion splitting rule is also called **assertion transformation rule** and is written like:

`Ai1 implies A .`

R_{QLOCK} (set of reachable states) of OTS_{QLOCK} (OTS defined by the module QLOCK)

Signature determining R_{QLOCK} = Sys

```
-- initial state
  op init : -> Sys {constr}
-- actions
  bop want : Sys Pid -> Sys {constr}
  bop try  : Sys Pid -> Sys {constr}
  bop exit : Sys Pid -> Sys {constr}
```

Recursive definition of R_{QLOCK}

$$
\begin{aligned}
R_{QLOCK} = \mathtt{Sys} = \{\mathtt{init}\} \cup \\
&\{\mathtt{want}(s,i) \mid s \in \mathtt{Sys}, i \in \mathtt{Pid}\} \cup \\
&\{\mathtt{try}(s,i) \mid s \in \mathtt{Sys}, i \in \mathtt{Pid}\} \cup \\
&\{\mathtt{exit}(s,i) \mid s \in \mathtt{Sys}, i \in \mathtt{Pid}\}
\end{aligned}
$$

Induction Scheme (I.S.) induced by the structure of R_{QLOCK} = Sys

$$\texttt{mx}(s) =_{\text{def}} \forall i,j \in \texttt{Pid}.\texttt{inv1}(s,i,j)$$

$$
\begin{array}{l}
\{ \quad (\texttt{INV1} \models \texttt{mx(init)}), \\
(\texttt{INV1} \cup \{\texttt{mx}(s)\texttt{=true}\} \models \forall k \in \texttt{Pid}.\texttt{mx}(\texttt{want}(s,k))), \\
(\texttt{INV1} \cup \{\texttt{mx}(s)\texttt{=true}\} \models \forall k \in \texttt{Pid}.\texttt{mx}(\texttt{try}(s,k))), \\
(\texttt{INV1} \cup \{\texttt{mx}(s)\texttt{=true}\} \models \forall k \in \texttt{Pid}.\texttt{mx}(\texttt{exit}(s,k))) \ \} \\
\qquad \texttt{implies} \\
(\texttt{INV1} \models \forall s \in \texttt{Sys}.\texttt{mx}(s))
\end{array}
$$

Induction Scheme (Assertion Splitting via I.S.)

```
{[1-init],[1-want]*,[1-try]*,[1-exit]*}
            implies [mx]*
```

Assertion Splitting via Case Splitting

proof-02.cafe

Because

```
INV1 |- c-want(s,k) or ~c-want(s,k)
```

Holds, the following assertion splitting is justified.

Assertion Splitting via Case Splitting

```
{[1-want,c-w]*, [1-want,~c-w]}
        implies [1-want]*
```

(CS) $\{ (E \models (p_1 \text{ or } p_2)), (E \cup \{p_1 = true\} \models p), (E \cup \{p_2 = true\} \models p) \}$ implies $E \models p$

Changed!

Meta Level Equation and Object Level Equation

(EQ) $E \cup \{ t_1 = t_2 \} \models p$ iff $E \cup \{ (t1 = t2) = \text{true} \} \models p$

$(p = \text{true})$ iff p

```
--> [1-want,c-w-org]*
open INV1
 op s : -> Sys .
 ops i j k : -> Pid .
 eq inv1(s,I:Pid,J:Pid) = true .
 eq c-want(s,k) = true .
-- |=
 red inv1(want(s,k),i,j) .
close
```

as assertion ⟷ iff

as p.p. ⟹ EQ

```
--> [1-want,c-w]*
open INV1
 op s : -> Sys .
 ops i j k : -> Pid .
 eq inv1(s,I:Pid,J:Pid) = true .
 -- eq c-want(s,k) = true .
 eq pc(s,k) = rm .
-- |=
 red inv1(want(s,k),i,j) .
close
```

Assertion Transformation: INST,TRANS,HIDE,IMP

proof-04.cafe

The proof passage `[1-want,c-w,~i=k,~j=k]*` returns a Boolean term witch is equivalent to `inv1(s,i,j)`. This should return `true` because the Boolean term is an instance of `inv1 (s,I:Pid,J:Pid)` which is declared in the premise part (the part before `|=`) of this proof passage. This assertion/proof-passage is transformed to the effective one (the one which return true) by using INST, TRANS, and HIDE transforming rules.

INST

```
-- [1-want,c-w,~i=k,~j=k]*
open INV1
   op s : -> Sys . ops i j k : -> Pid .
   eq inv1(s,I:Pid,J:Pid) = true .
   -- eq c-want(s,k) = true .
   eq pc(s,k) = rm .
   eq (i = k) = false .
   eq (j = k) = false .
-- |=
   red inv1(want(s,k),i,j) .
close
```

iff

INST

```
-- [1-want,c-w,~i=k,~j=k,inst]*
open INV1
   op s : -> Sys . ops i j k : -> Pid .
   eq inv1(s,I:Pid,J:Pid) = true .
   eq inv1(s,i,j) = true .   **
   -- eq c-want(s,k) = true .
   eq pc(s,k) = rm .
   eq (i = k) = false .
   eq (j = k) = false .
-- |=
   red inv1(want(s,k),i,j) .
close
```

TRANS, HIDE

```
-- [1-want,c-w,~i=k,~j=k,inst,trans]*
open INV1
   op s : -> Sys . ops i j k : -> Pid .
   eq inv1(s,I:Pid,J:Pid) = true .
   -- eq c-want(s,k) = true .
   eq pc(s,k) = rm .
   eq (i = k) = false .
   eq (j = k) = false .
-- |=
   red inv1(s,i,j) implies
         inv1(want(s,k),i,j) .   **
close
```

iff

TRANS

including comments

iff

Comment-out-ed equation is effective as "assertion"

```
-- [1-want,c-w,~i=k,~j=k,inst,trans,hide]
open INV1
   op s : -> Sys . ops i j k : -> Pid .
   -- eq inv1(s,I:Pid,J:Pid) = true .   **
   -- eq c-want(s,k) = true .
   eq pc(s,k) = rm .
   eq (i = k) = false .
   eq (j = k) = false .
-- |=
   red inv1(s,i,j) implies
         inv1(want(s,k),i,j) .
close
```

excluding comments

implies

HIDE

Some basic properties of E |- p (1)

(term or equation which moves across |- is assumed to include no variables)

Let t1' and t2' be the terms obtained from terms t1 and t2 by replacing variables in t1 and t2 with corresponding ground terms respectively then:

(INST) (E U {t1=t2})|= p iff (E U {t1=t2} U {t1'=t2'})|= p

(TRANS) E |= ((t_1 = t_2) implies p) iff E U { t_1 = t_2 } |= p

Some basic properties of E |- p (2)
(term or equation which moves across |- is assumed to include no variables)

(HIDE) (E U {t1=t2})|= p implies
(E U {t1=t2} U {t3=t4})|= p

This justifies to comment out any equation (removing by making it comment) at any moment. (as a p.-p.)

(IMP) E |= (t_1 = t_2) implies
(E U { t_1 = t_2 } |= p iff E |- p)

module ISTEP1

invarients-1.cafe, proof-05.cafe->proof-06.cafe

```
red inv1(s,i,j) implies inv1(want(s,k),i,j) .
```

```
eq istep1(I:Pid,J:Pid) =
   inv1(s,I,J) implies inv1(s',I,J) .
```

red `istep1(i,j).`

Notice that using INST,TRANS,HIDE, and `istep1,`

```
ops k l : -> Pid .   …
-- |=
red inv1(s,k,l) implies istep1(i,j) .
```

can also be used instead of

```
-- |=
istep1(i,j)
```

for any `k` and `l.`

Simultaneous Induction Scheme

simultaniousIS.txt,invariants-2.cafe, proof-09.cafe->proof-10.cafe

Lemma Discovery/Introduction:
Invariant `inv2` is decided to be a lemma to be introduced.

Simultaneous Induction Scheme

```
{ [1-init], [1-2-want]*, [1-2-try]*, [1-2-exit]*,
  [2-init], [2-2-want]*, [2-2-try]*, [2-2-exit]* }
   implies [mx]*
and
{ [1-init], [1-2-want]*, [1-2-try]*, [1-2-exit]*,
  [2-init], [2-2-want]*, [2-2-try]*, [2-2-exit]* }
   implies [inv2]*
```

Lemma Usage: **Invariant predicate `inv2` can be declared in the premise part of any assertion/proof-passage after the introduction.**

```
1-init
1-want,c-w,i=k
1-want,c-w,~(i=k),j=k
1-want,c-w,~(i=k),~(j=k),istep1
1-want,~c-w
1-try,c-t,i=k,j=k
1-try,c-t,i=k,~(j=k)*
1-try,c-t,~(i=k)*
1-try,~c-t
1-exit*
```

```
1-init
1-want,c-w,i=k
1-want,c-w,~(i=k),j=k
1-want,c-w,~(i=k),~(j=k),istep1
1-want,~c-w
1-try,c-t,i=k,j=k
1-2-try,c-t,i=k,~(j=k),inv2
1-try,c-t,~(i=k)*
1-try,~c-t
1-exit*
2-init
2-2-want*
2-2-try*
2-2-exit*
```

Lemma declaration and its usage

```
-- [1-try2,c-t,i=k,~j=k,inv2]
open INV2
   ops i j k : -> Pid .
   -- eq inv1(s,I:Pid,J:Pid) = true .
   -- eq inv2(s,J:Pid) = true .
   -- eq c-try(s,k) = true .
   eq pc(s,k) = wt .
   eq top(queue(s)) = k .
   eq i = k .
   eq (j = k) = false .
-- successor state
   eq s' = try(s,k) .
-- |=
   red inv2(s,j) implies istep1(i,j) .
close
```

declared lemma

used lemma

No need to change already constructed assertions/proof-passages by lemma introduction

```
-- [1-want]*
open INV1
   op s : -> Sys .
   ops i j k : -> Pid .
   eq inv1(s,I:Pid,J:Pid) = true .
-- |=
   red inv1(want(s,k),i,j) .
close
```

implies

```
-- [1-want2]*
open ISTEP2
   ops i j k : -> Pid .
   -- eq inv1(s,I:Pid,J:Pid) = true .
   -- eq inv2(s,I:Pid) = true .
   eq s' = want(s,k) .
-- |=
   red istep1(i) .
close
```

Final Proof Score for QLOCK

[1-init]
[1-want,c-w,i=k]
[1-want,c-w,~i=k,j=k]
[1-want,c-w,~i=k,~j=k,istep1]
[1-want,~c-w]
[1-try,c-t,i=k,j=k]
[1-2-try,c-t,i=k,~j=k,inv2]
[1-2-try,c-t,~i=k,j=k,inv2]
[1-2-try,c-t,~i=k,~j=k]
[1-try,~c-t]
[1-2-exit,c-e,i=k]
[1-2-exit,c-e,~i=k,j=k]
[1-2-exit,c-e,~i=k,~j=k]
[1-2-exit,~c-e]

[2-init]
[2-2-want,c-w,i=k]
[2-2-want,c-w,~i=k,queue(s)=empty]
[2-2-want,c-w,~i=k,queue(s)=j,q]
[2-2-want,~c-w]
[2-2-try,c-t,i=k]
[2-2-try,c-t,~i=k]
[2-2-try,~c-t]
[2-2-exit,c-e,,i=k]
[2-2-exit,c-e,~i=k,pc(s,i)=cs,inv1]
[2-2-exit,c-e,~i=k,~pc(s,i)=cs]
[2-2-exit,~c-e]